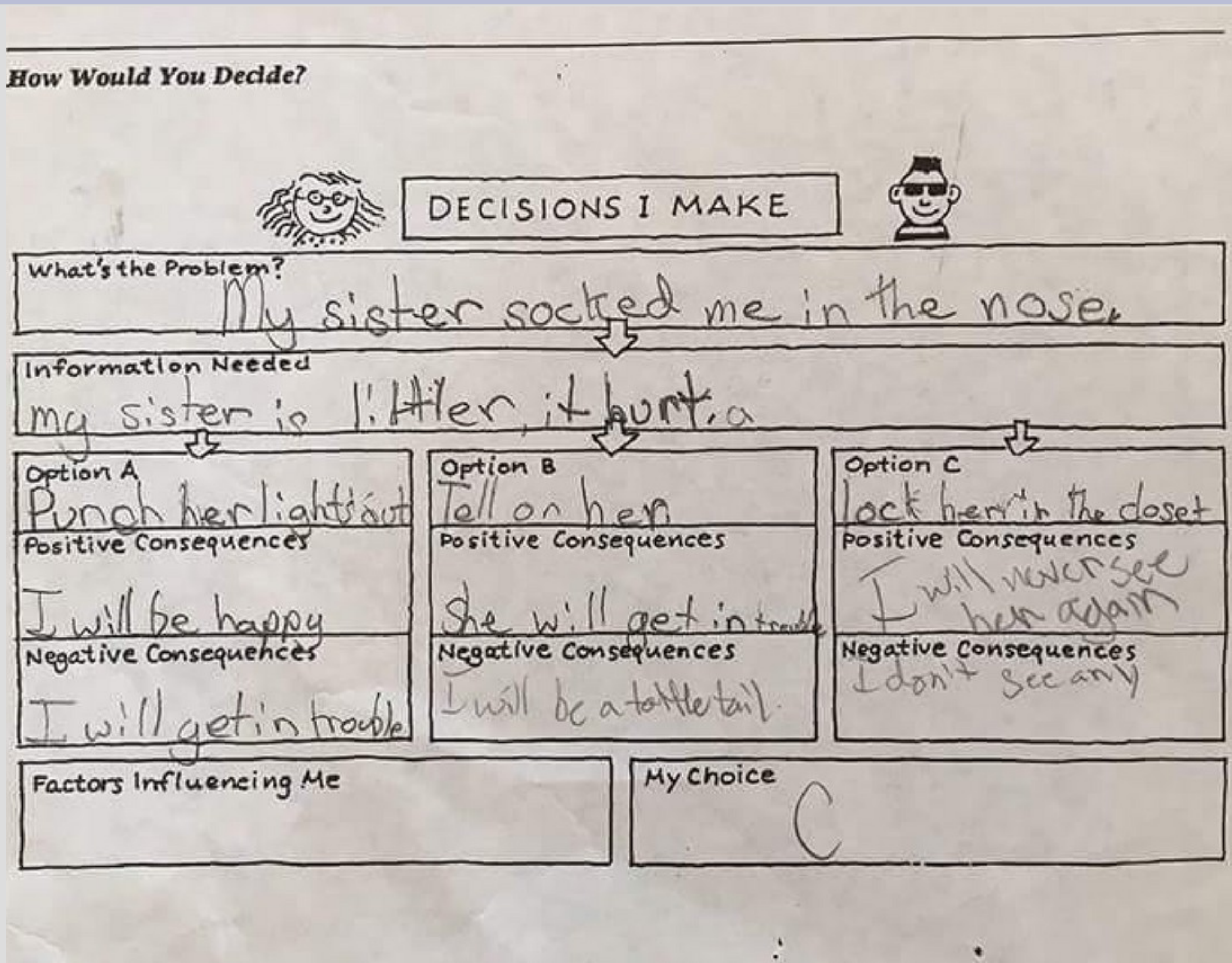
Decision Trees (Ch. 18.1-18.3)


How Would You Decide?

DECISIONS I MAKE

What's the Problem?
My sister socked me in the nose.

Information Needed
my sister is littler, it hurt.

Option A	Option B	Option C
Punch her lights out	Tell on her	lock her in the closet
Positive Consequences	**Positive Consequences**	**Positive Consequences**
I will be happy	She will get in trouble	I will never see her again
Negative Consequences	**Negative Consequences**	**Negative Consequences**
I will get in trouble	I will be a tattle tail.	I don't see any

Factors Influencing Me

My Choice
C

Learning

We will (finally) move away from uncertainty (for a bit) and instead focus on learning

Learning algorithms benefit from flexibility to solver a wide range of problems, especially:

(1) Cannot explicitly program (what set of if-statements/loops tells dogs from cats?)

(2) Answers might change over time (what is “trendy” right now?)

Learning

We can categorize learning into three types:

Unsupervised = No explicit feedback

Reinforcement = Get a reward or penalty based on quality of answer

Supervised = Have a set of inputs with the correct answer/output (“labeled data”)

Learning

Horse:

House:

Unsupervised:

Computer guesses: “donkey”, “spaceship”
You don’t tell it anything: “...”

Learning

Horse:

House:

Reinforcement:

Computer guesses: “donkey”, “spaceship”

You answer: “close”, “not really”

Learning

Horse:

House:

Supervised:

Computer doesn’t guess: “...”

You tell: “that is horse”, “that is house”

Learning

We can categorize learning into three types:

Unsupervised = No explicit feedback

Reinforcement = Get a reward or penalty based on quality of answer

easiest... so we will assume this for a while

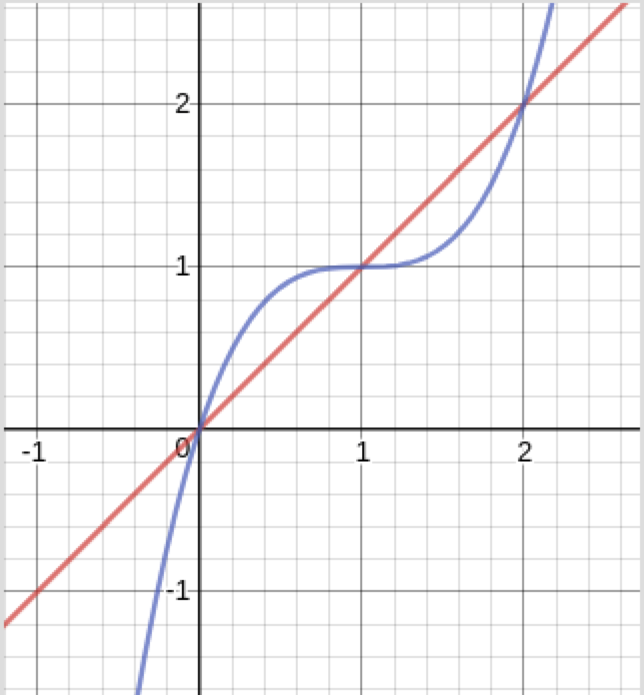
Supervised = Have a set of inputs with the correct answer/output ("labeled data")

Learning Trade-offs

One import rule is Ockham's razor which is:
if two options work equally well, pick simpler

For example, assume we want to find/learn
a line that passes through:
(0,0), (1,1), (2,2)

Quite obviously "y=x" works,
but so does "$y=x^3-3x^2+3x$"
... "y=x" is a better choice



Learning Trade-offs

A similar (but not same) issue that we often face in learning is overfitting

This is when you try too hard to match your data and lose a picture of the "general" pattern

This is especially important if noise or errors are present in the data we use to learn (called training data)

Learning Trade-offs

A simple example is suppose you want a line that passes through more points:
(0,0), (1,1), (2,2), (3,3), (4,4), (5,5.1), (6,6)

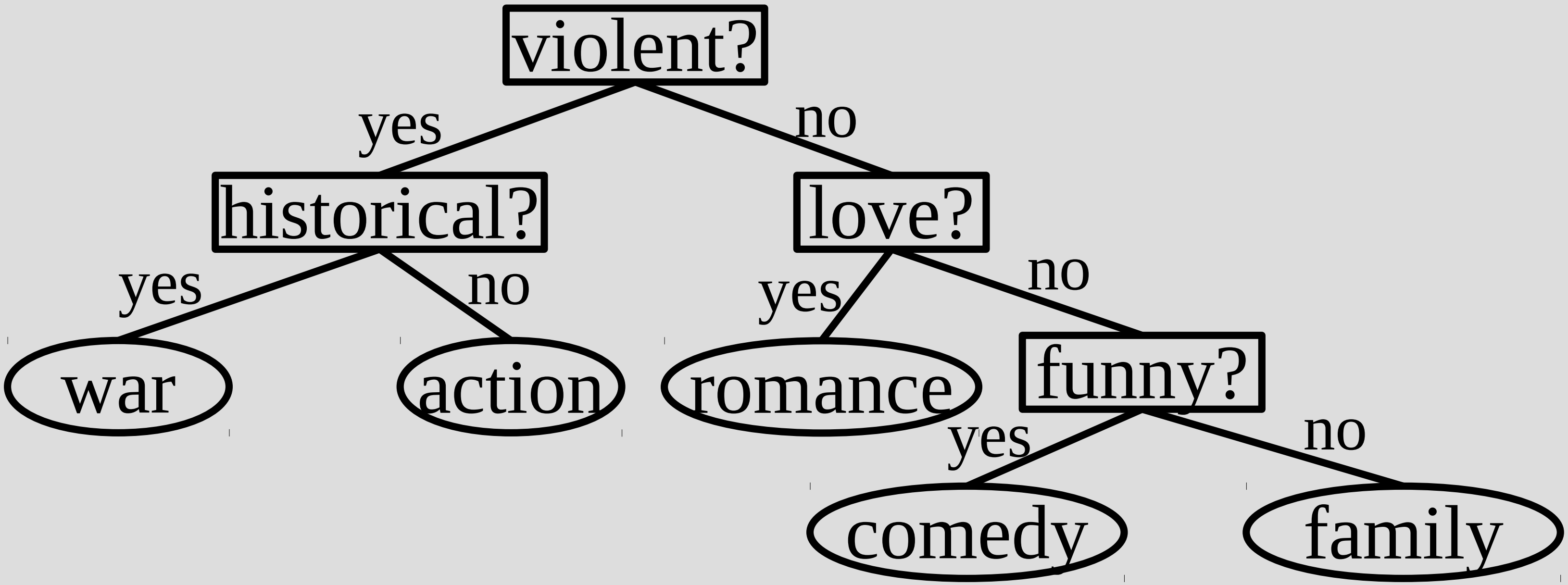
Line “y=x” does not quite work due to (5,5.1)

But it might not be worth using a degree 6 polynomial (not because finding one is hard), as it will “wiggle” a lot, so if we asked for y when x=10... it will be huge (or very negative)

Decision Trees

One of the simplest ways of learning is a <u>decision tree</u> (i.e. a flowchart... but no loops)

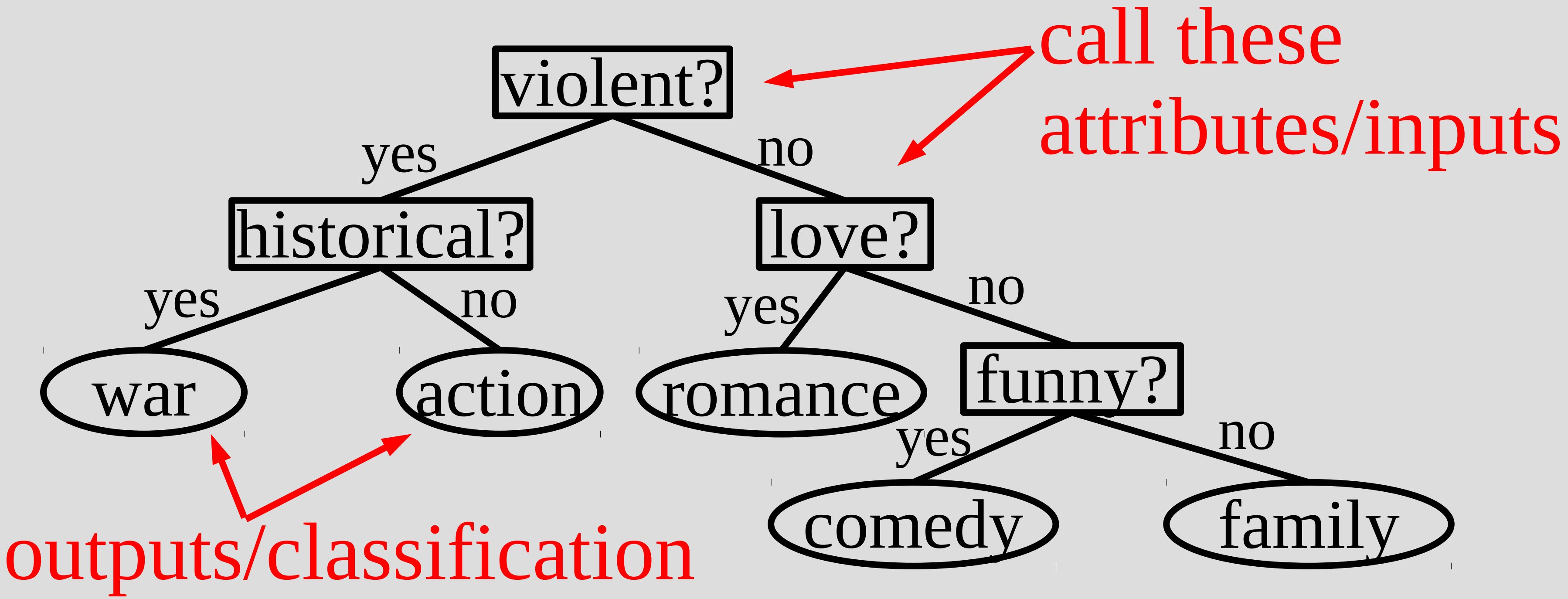
For example, you could classify movies as:



Decision Trees

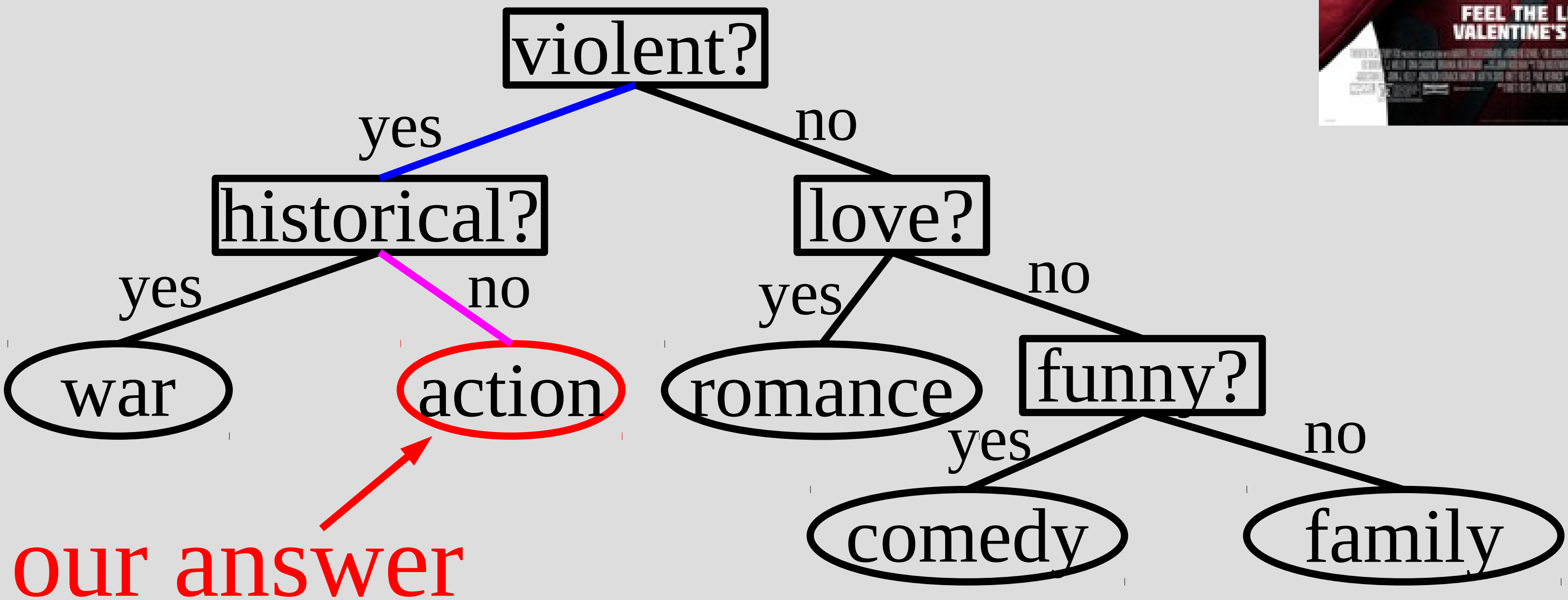
One of the simplest ways of learning is a <u>decision tree</u> (i.e. a flowchart... but no loops)

For example, you could classify movies as:



Decision Trees

If I wanted to classify Deadpool our inputs might be: [violent=yes, historical=no, love=not really, funny=yes]



Decision Trees

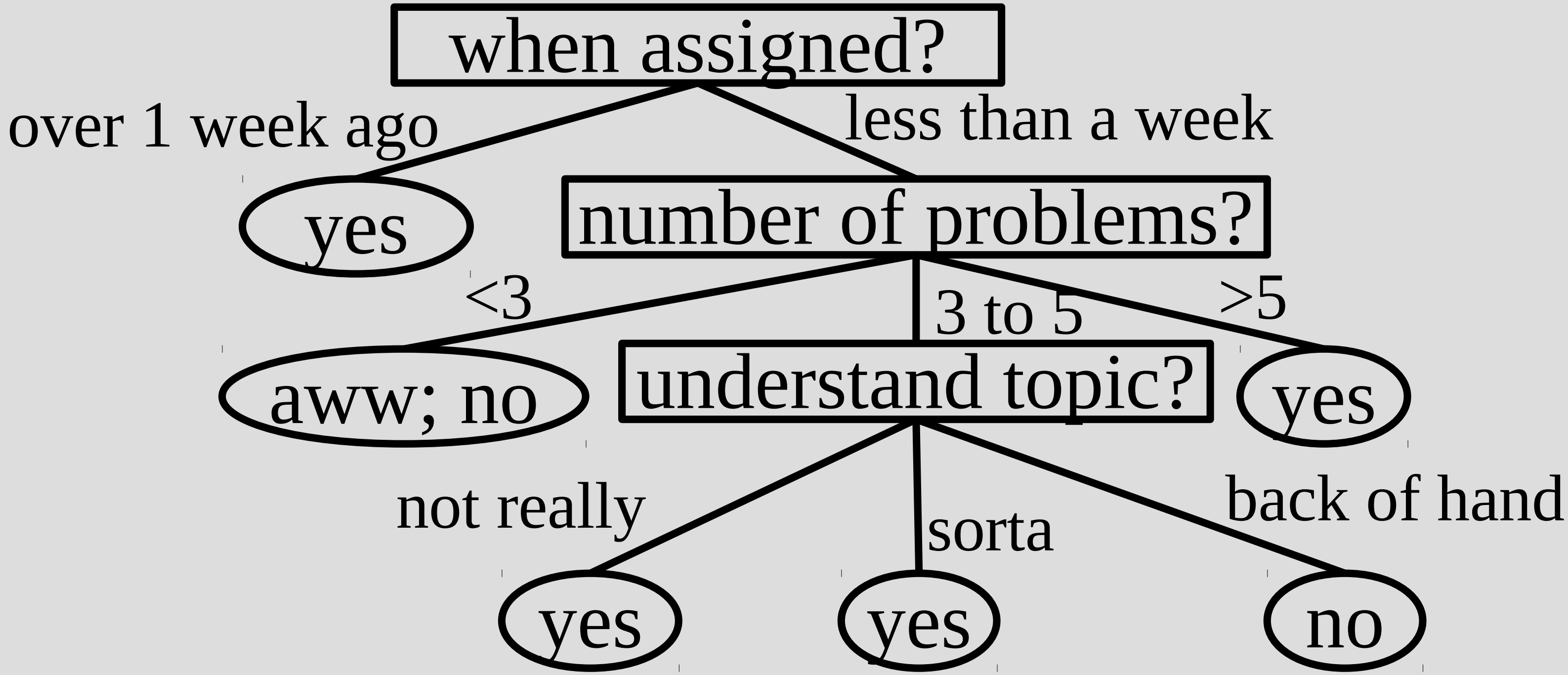
In our previous example, the attributes/inputs were binary (T/F) and output multivariate

The math is it simpler the other way around, input=multivariate & output=binary

An example of this might be deciding on whether or not you should start your homework early or not

Decision Trees

Do homework early example:



Making Trees

... but how do you **make** a tree from data?

Example	A	B	C	D	E	Ans
1	T	low	big	twit	5	T
2	T	low	small	FB	8	T
3	F	med	small	FB	2	F
4	T	high	big	snap	3	T
5	T	high	small	goog	5	F
6	F	med	big	snap	1	F
7	T	low	big	goog	9	T
8	F	high	big	goog	7	T
9	T	med	small	twit	2	F
10	F	high	small	goog	4	F

Making Tress: Brute Force

The brute force (try every option; find best) way would be: let n = 5 = number attributes

If these were all T/F attributes... there would be $2^n=2^5$ rows for a full truth table

Example	A	B	C	D	E	Ans
1	T	low	big	twit	5	T
2	T	low	small	FB	8	T
3	F	med	small	FB	2	F
4	T	high	big	snap	3	T
5	T	high	small	goog	5	F
6	F	med	big	snap	1	F
7	T	low	big	goog	9	T
8	F	high	big	goog	7	T
9	T	med	small	twit	2	F
10	F	high	small	goog	4	F

Making Tress: Brute Force

But each row of the truth table could be T/F

So the number of T/F combinations in the answer is:

$$2^{rows} = 2^{2^n}$$

This is very gross, so brute force is out

Example	A	B	C	D	E	Ans
1	T	low	big	twit	5	T
2	T	low	small	FB	8	T
3	F	med	small	FB	2	F
4	T	high	big	snap	3	T
5	T	high	small	goog	5	F
6	F	med	big	snap	1	F
7	T	low	big	goog	9	T
8	F	high	big	goog	7	T
9	T	med	small	twit	2	F
10	F	high	small	goog	4	F

Making Tress: Recursive

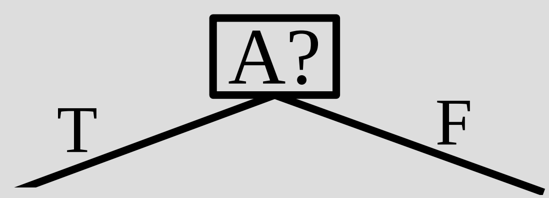
There are two key facts to notice:

(1) You need to pick an attribute to “split” on

(2) Then you have a recursive problem
  (1 less attribute, fewer examples)

Example	A	B	C	D	E	Ans
1	T	low	big	twit	5	T
2	T	low	small	FB	8	T
3	F	med	small	FB	2	F
4	T	high	big	snap	3	T
5	T	high	small	goog	5	F
6	F	med	big	snap	1	F
7	T	low	big	goog	9	T
8	F	high	big	goog	7	T
9	T	med	small	twit	2	F
10	F	high	small	goog	4	F

split A →



Example	B	C	D	E	Ans
1	low	big	twit	5	T
2	low	small	FB	8	T
4	high	big	snap	3	T
5	high	small	goog	5	F
7	low	big	goog	9	T
9	med	small	twit	2	F

Example	B	C	D	E	Ans
3	med	small	FB	2	F
6	med	big	snap	1	F
8	high	big	goog	7	T
10	high	small	goog	4	F

Making Tress: Recursive

This gives a fairly straight-forward recursive algorithm:

```
def makeTree(examples):
if output all T (or all F), make a leaf & stop
else    (1) A=pick attribute to split on
        for all values of A:
            (2) makeTree(examples with A val)
```

Making Tress: Recursive

What attribute should you split on?

Does it matter?

If so, what properties do you want?

Making Tress: Recursive

What attribute should you split on?

A very difficult question, the best answer is intractable so we will approximate

Does it matter?

Yes, quite a bit!

If so, what properties do you want?

We want a variable that separates the trues from falses as much as possible

Entropy

To determine which node to use, we will do what CSci people are best at:
copy-paste someone else's hard work

Specifically, we will "borrow" ideas from information theory about entropy
(which, in turn, is a term information theory "borrowed" from physics)

Entropy means a measure of disorder/chaos

Entropy

You can think of entropy as the number of "bits" needed to represent a problem/outcome

For example, if you flipped a fair coin...
you get heads/tails 50/50

You need to remember both numbers (equally)
so you need 1 bit (0 or 1) for both possibilities

Entropy

If you rolled a 4-sided die, you would need to remember 4 numbers (1, 2, 3, 4) = 2 bits

A 6-sided die would be $\log_2(6) = 2.585$ bits

If the probabilities are not uniform, the system is less chaotic... (fewer bits to "store" results)

So a coin always lands heads up: $\log_2(1) = 0$

Entropy

Since a 50/50 coin = 1 entropy/bits
... and a 100/0 coin = 0 entropy/bits

Then a 80/20 coin = between 0 and 1 bits

The formal formula is entropy, H(V), is:

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

... where V is a random variable and v_k is one entry in V (only uses prob, not value part)

Entropy

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

... so a 50/50 coin is random variable:

x = [(0.5, heads), (0.5, tails)]

$$H(x) = -0.5 \cdot \log_2(0.5) - 0.5 \cdot \log_2(0.5) = 1$$

Then... for our other examples:

y = [(0.8, heads), (0.2, tails)]

$$H(y) = -0.8 \cdot \log_2(0.8) - 0.2 \cdot \log_2(0.2) = 0.7219$$

z = [(1/6, 1), (1/6, 2), (1/6, 3), ... (1/6, 6)]

$$\begin{aligned} H(z) &= 6 \cdot (-\frac{1}{6} \cdot \log_2(\frac{1}{6})) \\ &= -\log_2(\frac{1}{6}) = \log_2(6) = 2.585 \end{aligned}$$

Entropy

How can we use entropy to find good splits?

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

Entropy

How can we use entropy to find good splits?

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

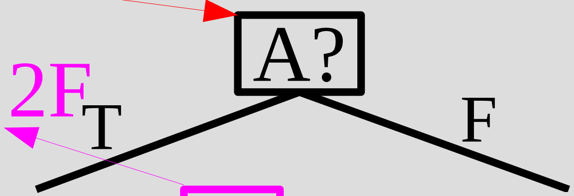
Compare entropy/disorder before and after split:

before: 5 T, 5 F

Example	A	B	C	D	E	Ans
1	T	low	big	twit	5	T
2	T	low	small	FB	8	T
3	F	med	small	FB	2	F
4	T	high	big	snap	3	T
5	T	high	small	goog	5	F
6	F	med	big	snap	1	F
7	T	low	big	goog	9	T
8	F	high	big	goog	7	T
9	T	med	small	twit	2	F
10	F	high	small	goog	4	F

split A →

move info here



4 T, 2F

Example	B	C	D	E	Ans
1	low	big	twit	5	T
2	low	small	FB	8	T
4	high	big	snap	3	T
5	high	small	goog	5	F
7	low	big	goog	9	T
9	med	small	twit	2	F

1 T, 3 F

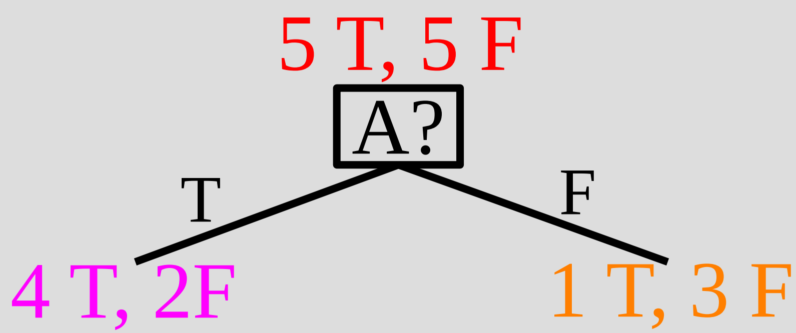
Example	B	C	D	E	Ans
3	med	small	FB	2	F
6	med	big	snap	1	F
8	high	big	goog	7	T
10	high	small	goog	4	F

Entropy

How can we use entropy to find good splits?

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

Compare entropy/disorder before and after split:



% of total true

$$H(before) = -0.5 \cdot \log_2(0.5) - 0.5 \cdot \log_2(0.5) = 1$$

$$H(after = T) = -0.667 \cdot \log_2(0.667) - 0.333 \cdot \log_2(0.333) = 0.918$$

$$H(after = F) = -0.25 \cdot \log_2(0.25) - 0.75 \cdot \log_2(0.75) = 0.811$$

Entropy

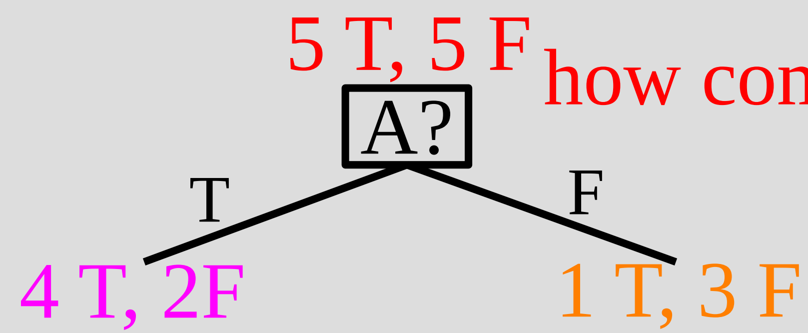
How can we use entropy to find good splits?

$$H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

Compare entropy/disorder before and after split:

% of total true

how combine?



$$H(before) = -0.5 \cdot \log_2(0.5) - 0.5 \cdot \log_2(0.5) = 1$$

$$H(after = T) = -0.667 \cdot \log_2(0.667) - 0.333 \cdot \log_2(0.333) = 0.918$$

$$H(after = F) = -0.25 \cdot \log_2(0.25) - 0.75 \cdot \log_2(0.75) = 0.811$$

Entropy

Random variables (of course)!

$after_A$ = [(6/10, 0.918), (4/10, 0.811)]

6 of 10 examples had A=T

So expected/average entropy after is:

$$E[after_A] = 0.6 \cdot 0.918 + 0.4 \cdot 0.811 = 0.875$$

We can then compute the difference (or gain):

$$Gain(A) = H(before) - H(after_A) = 1 - 0.875 = 0.125$$

More “gain” is means less disorder after

Entropy

So we can find the “gain” for each attribute and pick the argmax attribute

This greedy approach is not guaranteed to get the shallowest (best) tree, but does well

However, we might be over-fitting the data... but we can use entropy also determine this

Statistics Rant

Next we will do some statistics

\rantOn
Statistics is great at helping you make correct/accurate results

Consider this runtime data, is alg. A better?

A	5.2	6.4	3.5	4.8	3.6
B	5.8	7.0	2.8	5.1	4.0

Statistics Rant

Not really... only a 20.31% chance A is better (too few samples, difference small, var large)

A	5.2	6.4	3.5	4.8	3.6
B	5.8	7.0	2.8	5.1	4.0

Yet, A is faster 80% of the time... so you might be mislead in how great you think your algorithm is

\rantOff

Decision Tree Pruning

We can frame the problem as: what is the probability that this attribute just randomly classifies the result

Before our "A" split, we had with 5T and 5F
A=T had 4T and 2F

So 6/10 of our examples went A=T...
if these 6/10 randomly picked from the 5T/5F we should get 5*6/10 T on average randomly

Decision Tree Pruning

Formally, let p=before T=5, n=before false=5

$p_{A=T}$=T when "A=T" = 4

$n_{A=F}$=F when "A=T" = 2

... and similarly for $p_{A=F}$ and $n_{A=F}$

Then we compute the expected "random" outcomes:

$$\hat{p}_k = p \cdot \frac{p_k + n_k}{p + n} \qquad \hat{n}_k = n \cdot \frac{p_k + n_k}{p + n}$$

5 * 6/10 = 3 T on average by "luck"

Decision Tree Pruning

We then compute (a "test statistic"):

$$
\begin{aligned}
x &= \sum_k \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k} \\
&= \frac{(p_{A=T} - \hat{p}_{A=T})^2}{\hat{p}_{A=T}} + \frac{(n_{A=T} - \hat{n}_{A=T})^2}{\hat{n}_{A=T}} \\
&\qquad + \frac{(p_{A=F} - \hat{p}_{A=F})^2}{\hat{p}_{A=F}} + \frac{(n_{A=F} - \hat{n}_{A=F})^2}{\hat{n}_{A=F}} \\
&= \frac{(4-3)^2}{3} + \frac{(2-3)^2}{3} + \frac{(1-2)^2}{2} + \frac{(3-2)^2}{2} \\
&= 1.667
\end{aligned}
$$

Decision Tree Pruning

Once we have "x" we can jam it into the χ^2 (chi-squared) distribution:

$$\chi^2(1)(x) = \chi^2(1)(1.667) = 0.19671$$

= [possible attribute values] -1 (degrees of freedom)

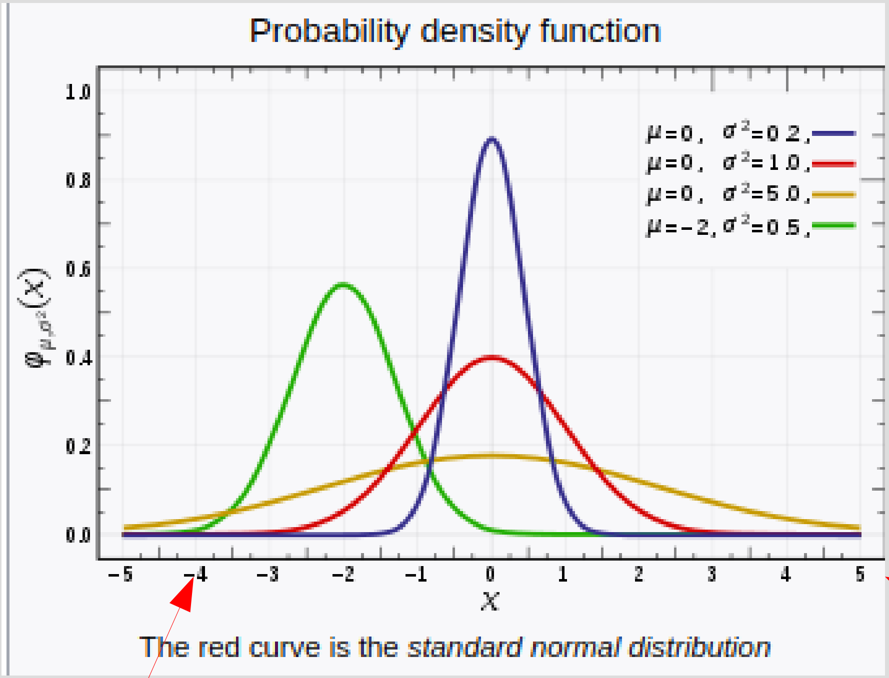
So there is a 19.67% chance this variable is just "randomly" assigning... so we might want to not use "A" here (other places maybe)

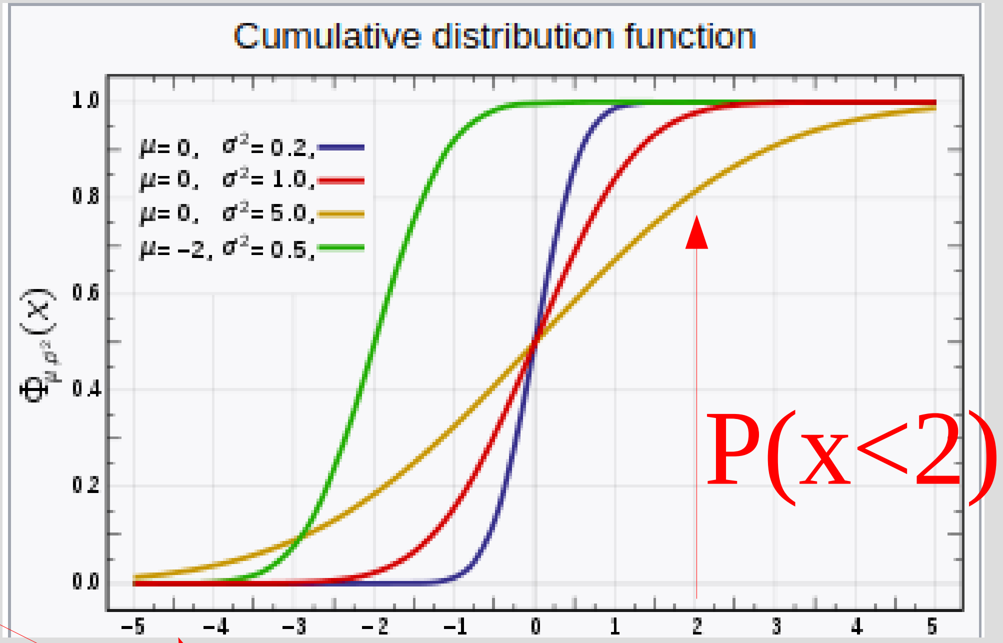
for T/F happens when x>3.841

The "typical" threshold we look for is 5% of being "random"... if so, could collapse node

What is this x^2 thing?

I think most people are familiar with the "bell"/normal/Gaussian distribution:





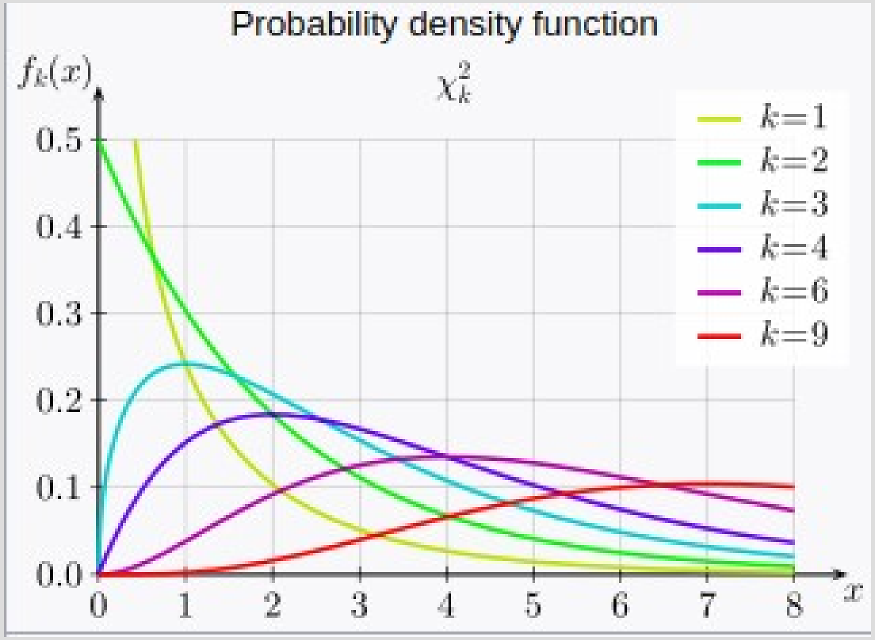
$N(\mu,\sigma^2)(x)$ needs 2 paramters: μ,σ

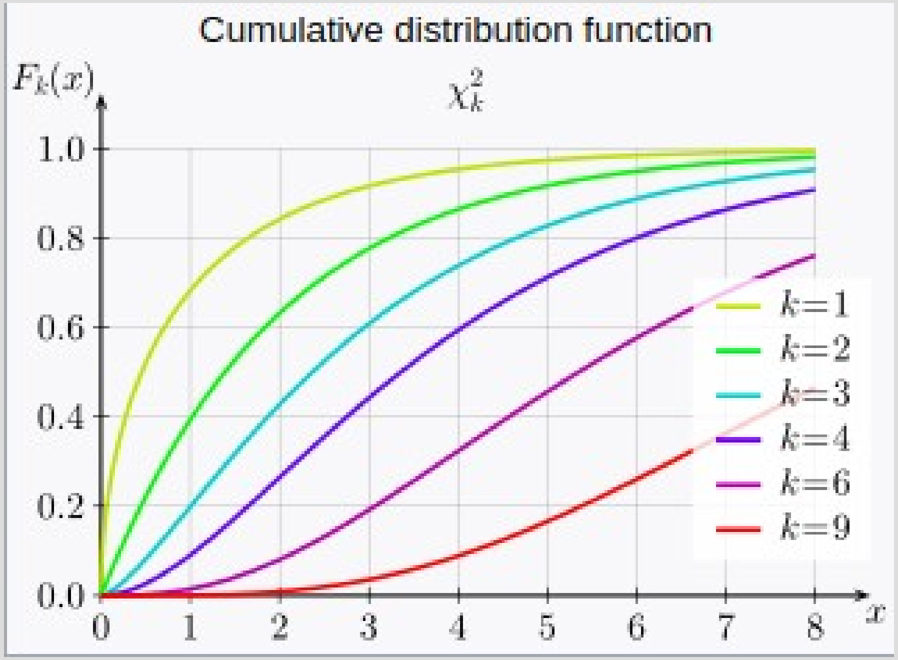
$$\int_{-\infty}^{x} P(z)dz = P(x < z)$$

What is this χ^2 thing?

a statistics thing... out of the scope of this course

χ^2 is just a different distribution that only requires 1 parameter (degrees of freedom)

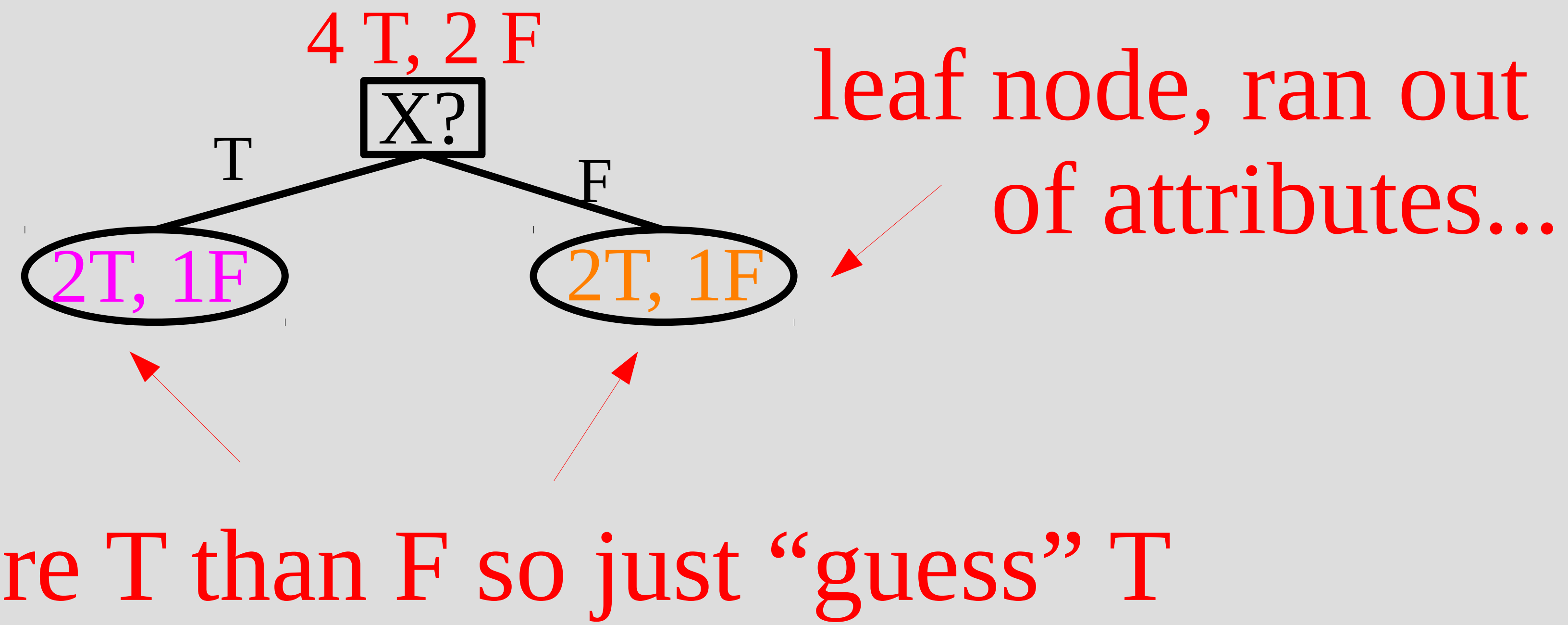




Written both as $\chi^2(k,x)$ or $\chi^2(k)(x)$

Decision Tree Pruning

So, suppose you had a “bad” attribute (conflicting examples/inputs in this case):



Notice the attribute “X” is not really helping (at all...), so you could just remove it

Complications

There are a number of complications:

(1) Attributes with more possible "values" seem better than they are

(2) Integers/doubles you typically want to threshold to remove issue of (1)

(3) If you want a continuous output rather than a classification, your leaf needs to be a function rather than a single value